Extensions – Introduction

- Similiar to Swift, C# and Gosu extensions in Kotlin expand the functionality of standard classes.
- **Kotlin** supports **extensions functions** as well as **extension properties**.

Extensions – Introduction

How to we declare extension functions ?

Just write a function as you would normally, and put the name of the class (receiver) before the function name seperated by a "."

```
fun View.visible() {
   this.visibility = View.VISIBLE
}
```

Receiver class (View)

Extensions – Rules – Members always win

Member funtions always win.

```
class C {
    fun foo() {
        println("member")
    }
}
```

```
fun C.foo() { println("extension") }
```

It will print „member“ always

Extensions – Rules – Nullable Extensions

It is possible to define extension functions for a nullable types

```
fun Any?.toString(): String {
   if (this == null) return "null"
   return toString()
}
```

Extensions – Rules – Static Extensions

You can define extension function on class level instead of instance level.

```
class Foo{

   companion object

   fun sayHello() = "Hello"

}
```

```
fun Foo.Companion.sayBye() = "Bye"
```

Extensions – Properties

As mentioned in the beginning, you can also define extension properties for classes. The only restriction is that you can‘t initialize the property directly. You have do explicitly define the getter and setter for it.

```
val Foo.bar = 1
```

// error: initializers are not allowed for extension properties

```
val Foo.bar: Int
    get() = 1
```

Extensions – Dispatching

```
interface Loggable {

    val Any.LOGGER: Logger
        get() = Logger.getLogger(javaClass.name)

    fun Any.logI(message: String){
        LOGGER.log(Level.INFO,message)
    }

    fun Any.logE(message: String, error: Throwable){
        LOGGER.log(Level.SEVERE,message,error)
    }
    //...
}
```

Dispatch Receiver

Extension Receiver

Extensions – Reified and Inline

```
inline fun <reified T : Activity> Activity.navigateTo(intentParameters: Map<String, Serializable>) {
   val intent = Intent(this, T::class.java)
   intentParameters.forEach { s: String, serializable: Serializable ->  intent.putExtra(s, serializable)}
   startActivity(intent)
}
```

```
navigateTo<SessionActivity>(mutableMapOf())
```

Extension- Exercise 01

There are 2 Util classes in our application. One is the DateUtil class which provides a method to print a given „long“ value in a readable form and a ActivityUtil class which provides 2 methods to simplify the navigation from Activity to fragments.

1.) Move the functionality of the DateUtil class in a proper extension class on the correct receiver
2.) Move the functionality of the ActivityUtil class in a dispatching extension so that only Activities which implements this dispatching interface can take advantage of the extension methods.
(Note: U need to refactor the NavigationController to be of a generic type, so you can make use of the interface constraints)

Excercise:
*Checkout branch **chapter_02_section_05_extension_exercise** and search for „chapter_02_section_05_extension_exercise»*

Extension- Excercise 02

Until now the RedditOverviewAdapter has to call certain notify mehtods when the list of items gets updated. Google release a utility class called DiffUtil a year ago which removes the need to manually call this notify methods.
You can find a small tutorial about the DiffUtil class in the following link:
https://medium.com/@iammert/using-diffutil-in-android-recyclerview-bdca8e4fbb00
https://developer.android.com/reference/android/support/v7/util/DiffUtil.html

This exercise is not about the DiffUtil itself, but rather how we can combine different concepts of Kotlin to simplify and minimize repeating calls.

To correctly solve this exercise you should make use of the following aspects:

- Delegate(s)
- Dispatching Extensions
- Lambda / Higher-Order functions

Note: This exercise is a little bit tricky. (You have to implement one new class and all other modifications should only happen inside the RedditOverviewAdapter)

Checkout branch ***chapter_02_section_06_advanced_extension_exercise***